Control of Dynamics System

Jungdam Won

Computer Science & Engineering

Seoul National Univ.

This lecture slides were created based on the lecture notes of “An Introduction to Mathematical Optimal Control Theory, Lawrence C. Evans, UC Berkely” and “An Introduction to Reinforcement Learning, David Silver, Univ. of College London”.

What is Control?
Why Do We Need It?

- What
  - Rougly speaking, control is a mechanism of changing the output of dynamical system so that it matches to our desirable output
- Why
  - When simulating dynamical systems, interesting phenomena usually occur momentarily for a short time, however, we want to see them all the time in CG
  - Reproducing some phenomena by nature requires to implement a control mechanism
    - E.g. Human/Animal movement

Examples of Control in CG

- Control of passive dynamics system by adding external forces
  - Rigid/Soft Body Simulation
  - Fluids
- Reproducing human/animal behaviors
  - Crowd simulation
  - Reproducing locomotion in the existence of external purtabation

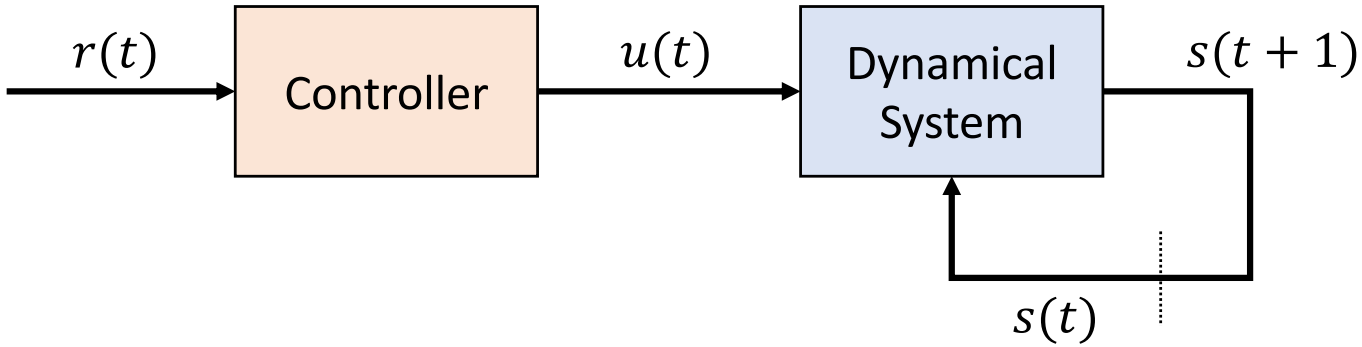
Agenda

- Open Loop Control
- Closed-Loop Control
  - PID Controller
- Optimal Control Theory
  - Shooting Methods
  - Model Predictive Control
  - (Deep) Reinforcement Learning

Agenda

- Open Loop Control
- Closed-Loop Control
  - PID Controller
- Optimal Control Theory
  - Shooting Methods
  - Model Predictive Control
  - (Deep) Reinforcement Learning

Open Loop Control



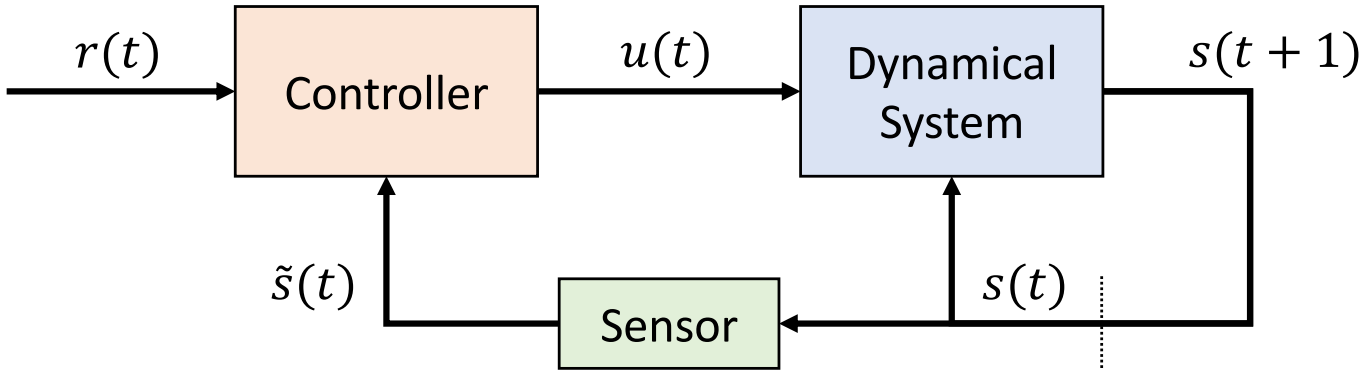
- An open loop controller is a controller in which control ouput is ***indepenent*** of the output of dynamical system
- If the desired state $r(t)$ is the same then the control oputput $u(t)$ at each timestep should the same
- E.g. boling eggs, toaster, etc

Open Loop Control

- Pros
  - Simple to implement, almost no computation is necessary in runtime
  - Convinenet if the system output is hard to measure

- Cons
  - The control strategy only works if the initial state of dynamical system is compatible with its fixed strategy
  - Unexpected external disturbance (i.e. noise) could mess up the result even if the initial state was compatible

Closed Loop (Feedback) Control



- The output $u(t)$ of a closed loop (feedback) controller ***depeneds*** on both the output of dynamical system $s(t)$ and the desired state $r(t)$
- Familiar examples are air conditioners, electric kettles, autonomous driving cars (it should!), and etc

Closed Loop (Feedback) Control

- Pros
  - The feedback mechanism enables the system to reach to the desired state even if the initial state is incompatible
  - It is resilent to external disturbance

- Cons
  - More complex to design than open-loop control
  - During runtime, it often requires more computation

An Example of Feedback Control: PID Control

- An error value $e(t)$ is computed as the difference between the reference output $r(t)$ and the system output $s(t)$

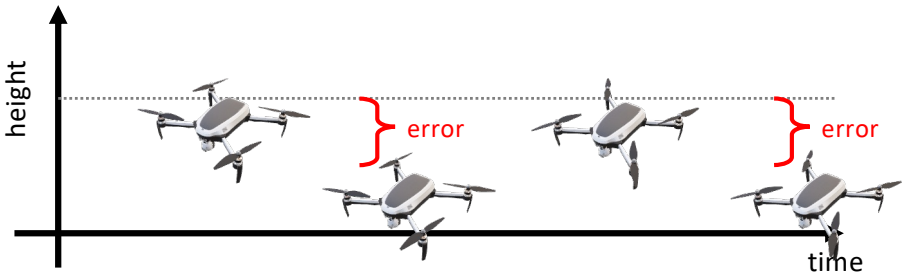
$$e(t) = r(t) - s(t)$$

- The control output $u(t)$ is computed based on **P**roportional, **I**ntegral, **D**erivative terms

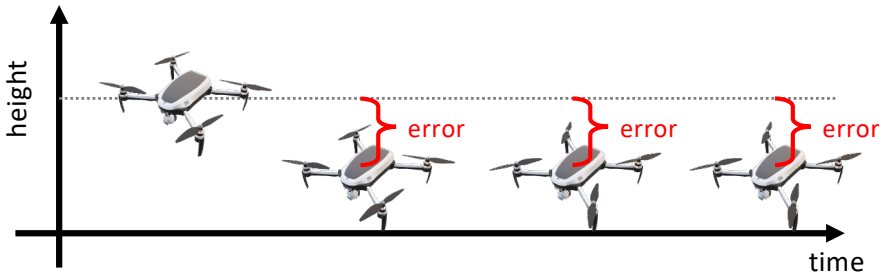
$$u(t) = k_P e(t) + k_I \int_{t-h}^{t} e(t) + k_D \dot{e}(t)$$

P Control

- **P**roportional term only
- If the current error is large, the control output will be proportionally large, which is controlled by k_P
- Using this term alone might result in an oscillatory or steady state error unless the dynamic system is stationary because it requires non-zero error, $e(t)$, to generate non-zero control output



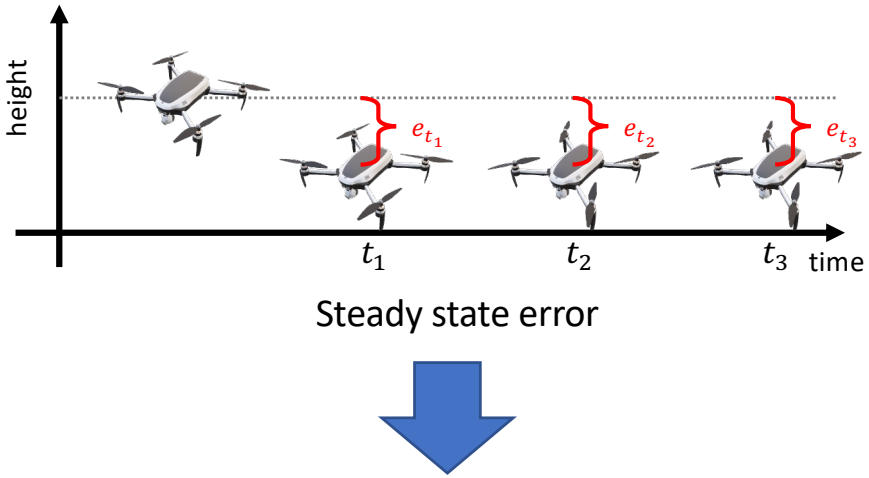
Oscillatory error



Steady state error

PI Control

- **P**roportional + **I**ntegral terms
- The integral term accounts for accumulated errors over time, so steady state error could be mitigated



$$u(t_3) = k_P e(t_3) + k_I \sum [e(t_1) + e(t_2) + e(t_3)]$$

PD Control

- **P**roportional + **D**erivative terms

- The derivative term would be one of the best estimates of the future (i.e. how the system will change soon), so it enables the controller quickly adapt to the change of the system

$$u(t) = k_P[r(t) - s(t)] + k_D[\dot{r}(t) - \dot{s}\,(t)]$$

PD Control

- The desired velocity $\dot{r}(t)$ can be approximated by the history of error

$$\dot{e}(t) = \dot{r}(t) - \dot{s}\,(t) \approx \frac{e(t) - e(t-1)}{dt}$$

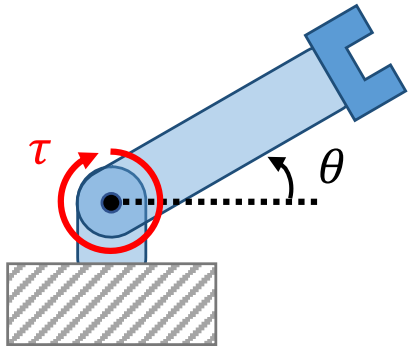
- Or it can set manually. If the desired velocity is always set as zero, the controller will smoothly adapt to the change of the system

$$u(t) = k_P[r(t) - s(t)] - k_D\dot{s}(t)$$

Tips for Parameter Tuning

1) Set all PID gains as zeros
2) Find the largest possible the proportional gain which does not incur the divergence of target dynamic system
3) If the steady state error is observed, increase the integral gain until the steady state error is removed. Otherwise, you could ignore this term
4) Tune the derivative gain
    1) When non-zero reference velocity is used (The goal is to increase the control response)
        1) If the system is too sensitive (easily diverging), consider decreasing the P gain and repeat from #2
    2) When zero reference veloicty is used (The goal is to increase the stability)
        1) If the system is too slow to response after adding this term, consider increasing the P gain and repeat from #2

A PD Controller for A Robot Arm



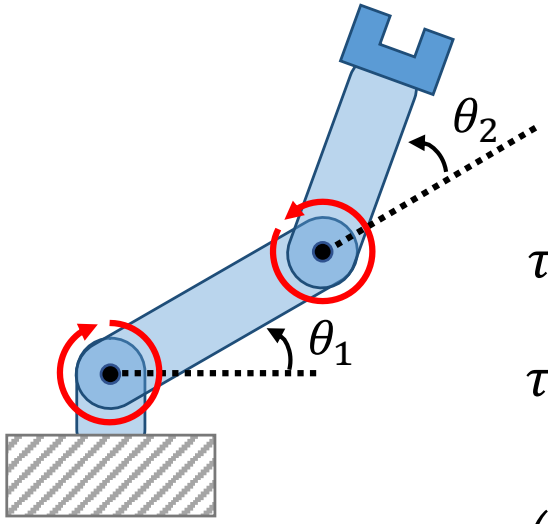
Error in joint angle

$$\tau(t) = k^P[\theta^r(t) - \theta(t)] + k^D[\dot{\theta}^r(t) - \dot{\theta}(t)]$$

Torque actuated

Error in joint angular vel.

A PD Controller for A Robot with Multiple Arms



$$\tau_1(t) = k_1^P[\theta_1^r(t) - \theta_1(t)] + k_1^D\left[\dot{\theta}_1^r(t) - \dot{\theta}_1(t)\right]$$

$$\tau_2(t) = k_2^P[\theta_2^r(t) - \theta_2(t)] + k_2^D\left[\dot{\theta}_2^r(t) - \dot{\theta}_2(t)\right]$$

$$\begin{pmatrix}\tau_1\\ \tau_2\end{pmatrix} = \begin{pmatrix}k_1^P & 0\\ 0 & k_2^P\end{pmatrix}\begin{pmatrix}e_1(t)\\ e_2(t)\end{pmatrix} + \begin{pmatrix}k_1^D & 0\\ 0 & k_2^D\end{pmatrix}\begin{pmatrix}\dot{e}_1(t)\\ \dot{e}_2(t)\end{pmatrix}$$

$$\boldsymbol{\tau} = K_P\boldsymbol{e}(t) + K_D\dot{\boldsymbol{e}}(t)$$

Would diagonal matrices be the best choices for K_P and K_D?

Agenda

- Open Loop Control
- Closed-Loop Control
  - PID Controller
- Optimal Control Theory
  - Shooting Methods
  - Model Predictive Control
  - (Deep) Reinforcement Learning

Optimal Control

- It is a branch of mathematical optimization that deals with finding ***a control for a dynamical system*** over a period of time such that an ***objective function is optimized*** *-wiki-*

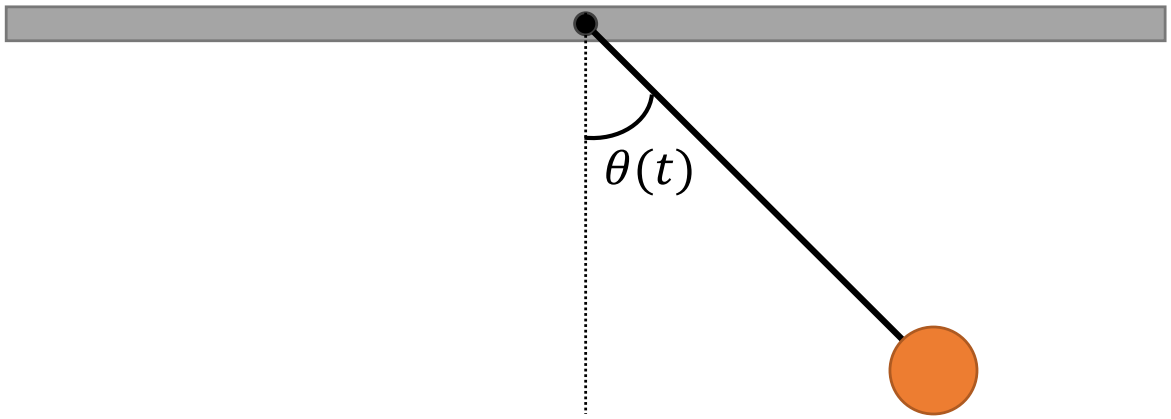
$$\mathcal{J}[\mathbf{u}(\cdot)] \equiv \int_0^T r\big(\mathbf{x}(t), \mathbf{u}(t)\big)dt + g(x(T))$$

$$\begin{cases} \dot{\mathbf{x}}(t) = f\big(\mathbf{x}(t), \mathbf{u}(t)\big) & (t > 0) \\ \mathbf{x}(0) = \mathbf{x}_0 \end{cases}$$: the system dynamics

- Our aim is to find an optimal control $\mathbf{u}^*(\cdot)$ which ***maximizes*** the total payoff (i.e. reward)

$$\mathbf{u}^*(\cdot) = \underset{\mathbf{u}(\cdot)}{\operatorname{argmax}}\, \mathcal{J}[\mathbf{u}(\cdot)]$$

Examples: A Pendulum



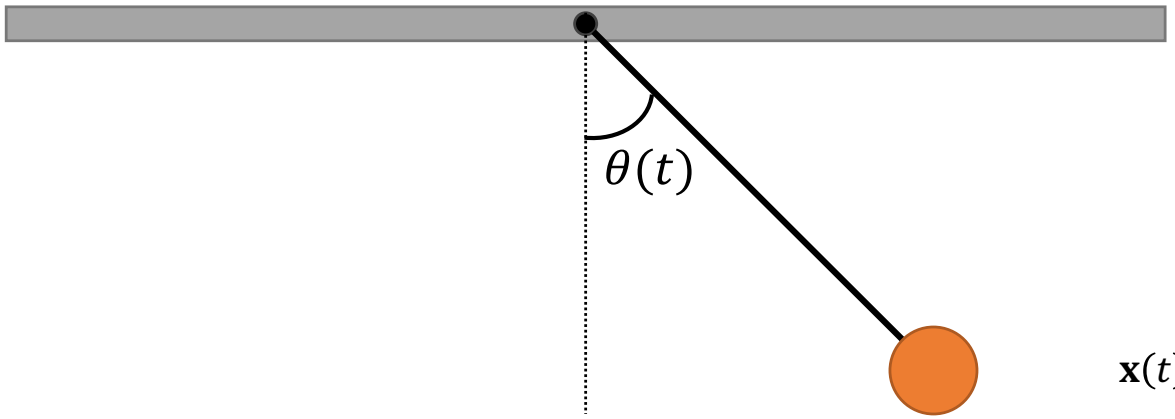
$$\ddot{\theta}(t) = -\lambda\dot{\theta}(t) - \omega^2\theta(t) + u(t)$$

$$\dot{\theta}(0) = \theta_1, \theta(0) = \theta_2$$

$$\mathbf{x}(t) = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} \theta(t) \\ \dot{\theta}(t) \end{pmatrix}$$

$$\begin{cases} \dot{\mathbf{x}}(t) = \begin{pmatrix} x_2(t) \\ -\lambda x_2(t) - \omega^2 x_1(t) + u(t) \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -\omega^2 & -\lambda \end{pmatrix} \mathbf{x}(t) + \begin{pmatrix} 0 \\ 1 \end{pmatrix} u(t) \\ \mathbf{x}(0) = \begin{pmatrix} \theta_2 \\ \theta_1 \end{pmatrix} \end{cases}$$

Examples: A Pendulum

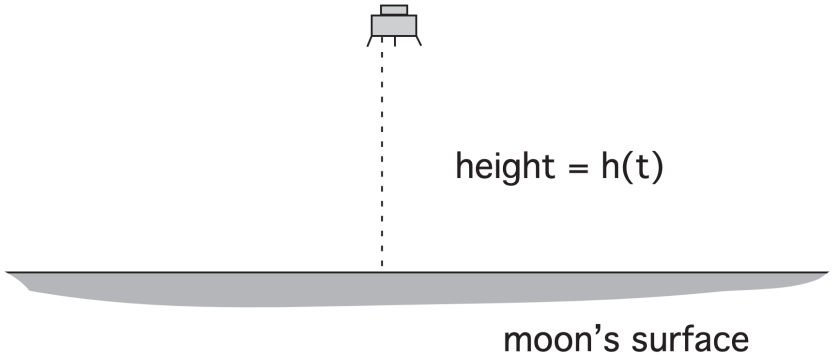


$$\mathbf{x}(t) = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} \theta(t) \\ \dot{\theta}(t) \end{pmatrix}$$

$$\mathcal{J}[\mathbf{u}(\cdot)] = -\int_0^T \mathbf{x}(t)^T \begin{pmatrix} k_1 & 0 \\ 0 & k_2 \end{pmatrix} \mathbf{x}(t) dt \qquad k_1 > 0, k_2 > 0$$

We are looking for a control sequence that brings the pendulum into the rest state $\left(\theta, \dot{\theta}\right)^T = (0,0)^T$ as quickly as possible

Examples: A Moon Lander



$h(t)$ = height at time t $\quad h(t) \geq 0$

$v(t)$ = velocity at time t

$m(t)$ = mass of spacecraft $\quad m(t) \geq 0$

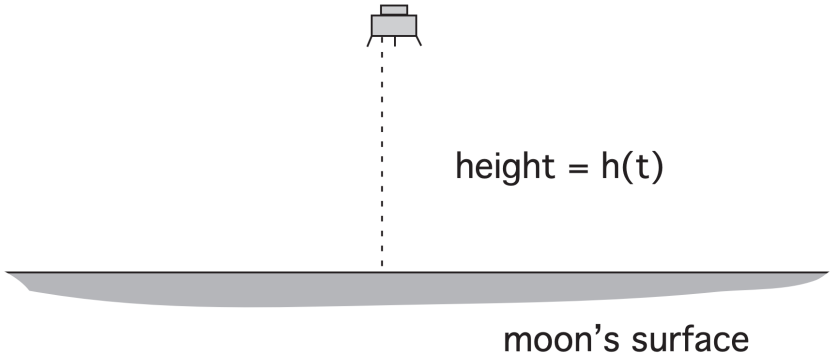
$u(t)$ = thrust at time t $\quad 0 \leq u(t) \leq 1$

$$\dot{m}(t) = -ku(t)$$
$$m(t)\ddot{h}(t) = -m(t)g + u(t)$$

$$\mathbf{x}(t) = \begin{pmatrix} h(t) \\ v(t) \\ m(t) \end{pmatrix}$$

$$\dot{\mathbf{x}}(t) = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \mathbf{x}(t) + \begin{pmatrix} 0 \\ 1/m(t) \\ -k \end{pmatrix} u(t) + \begin{pmatrix} 0 \\ -g \\ 0 \end{pmatrix}$$

Examples: A Moon Lander



$h(t) =$ height at time t $\quad h(t) \geq 0$

$v(t) =$ velocity at time t

$m(t) =$ mass of spacecraft $\quad m(t) \geq 0$

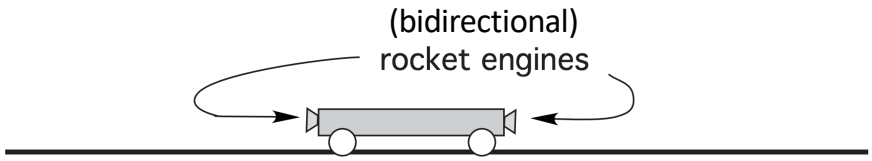
$u(t) =$ thrust at time t $\quad 0 \leq u(t) \leq 1$

$$\mathcal{J}[u(\cdot)] = -m(\tau)$$

τ denotes the first time that $h(\tau) = v(\tau) = 0$

- The goal is to bring a spacecraft to a soft landing on the lunar surface using the least amount of fuel
- This is a variable (open) endpoint problem, since the final time is not given in advance

Examples: A Rocket Railroad Car



$q(t) =$ position at time t

$v(t) =$ velocity at time t

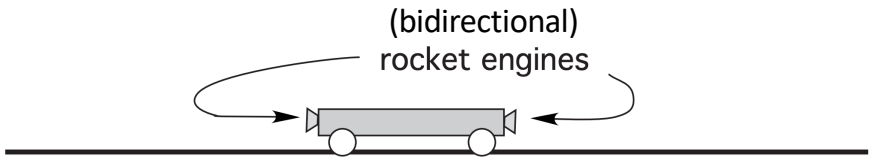
$u(t) =$ thrust from rockets $\quad -1 \leq u(t) \leq 1$

$$m\ddot{q}(t) = u(t)$$

$$\mathbf{x}(t) = \begin{pmatrix} q(t) \\ v(t) \end{pmatrix}$$

$$\begin{cases} \dot{\mathbf{x}}(t) = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \mathbf{x}(t) + \begin{pmatrix} 0 \\ 1/m \end{pmatrix} u(t) \\ \mathbf{x}(0) = (q_0, v_0) \end{cases}$$

Examples: A Rocket Railroad Car



$q(t) =$ position at time t

$v(t) =$ velocity at time t

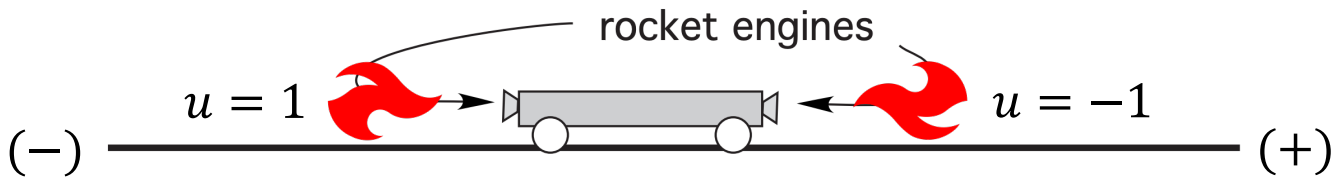
$u(t) =$ thrust from rockets $\quad -1 \leq u(t) \leq 1$

$$J[u(\cdot)] = -\int_0^\tau 1\, dt = -\tau$$

τ denotes the first time that $h(\tau) = v(\tau) = 0$

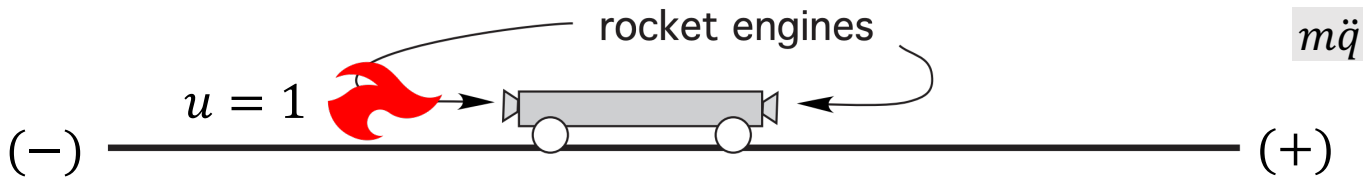
- To goal is to figure out how to fire the rockets, so as to arrive at the origin with zero velocity in a minimum amount of time
- This is a variable (open) endpoint problem, since the final time is not given in advance

A Rocket Railroad Car: A Geometric Solution



- We will focus our attention only upon those controls for which at each moment of time either the left or the right rocket engine is fired at full power (i.e. $u = 1$ or $u = -1$), and let's assume that the mass of the car is 1kg

A Rocket Railroad Car: A Geometric Solution



- **CASE 1**: $u = 1$ for some time interval

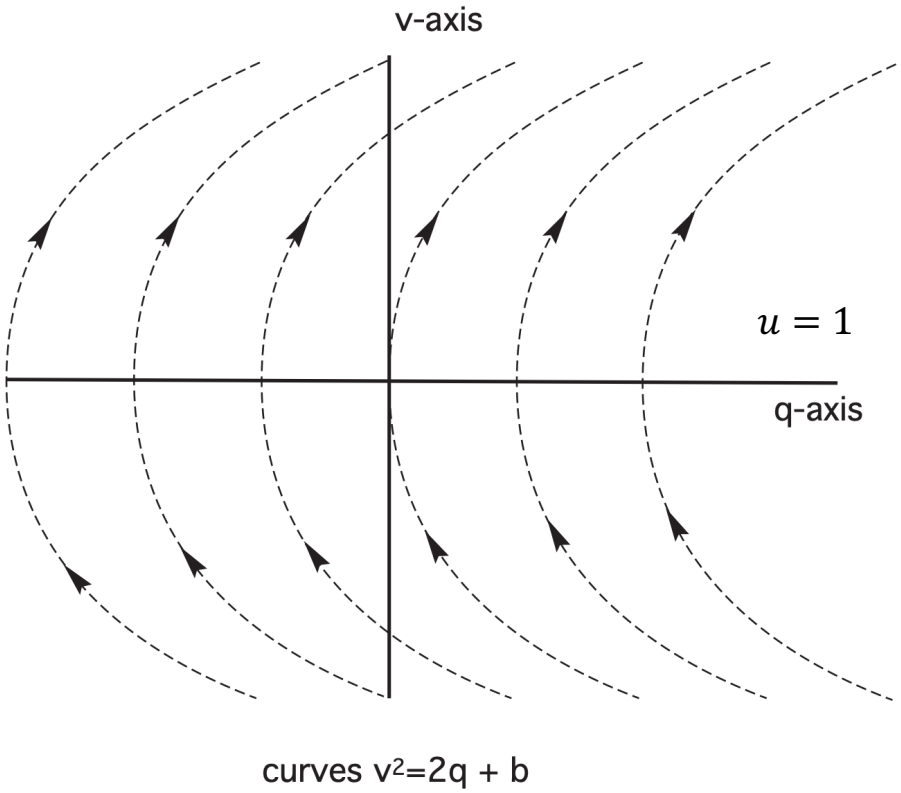
$$\begin{cases} \dot{q} = v \\ \dot{v} = 1 \end{cases} \Rightarrow v\dot{v} = \dot{q} \Rightarrow \frac{1}{2}(\dot{v^2}) = \dot{q}$$

- Let t_0 belong to the time interval where $u = 1$ and integrate from t_0 to t:

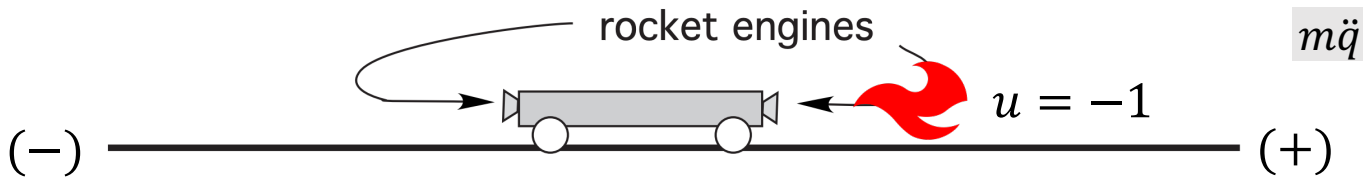
$$\frac{v^2(t)}{2} - \frac{v^2(t_0)}{2} = q(t) - q(t_0) \Rightarrow v^2(t) = 2q(t) + \underbrace{(v^2(t_0) - 2q(t_0))}_{b}$$

A Rocket Railroad Car: A Geometric Solution

- **CASE 1**: $u = 1$ for some time interval



A Rocket Railroad Car: A Geometric Solution



- **CASE 2**: $u = -1$ for some time interval

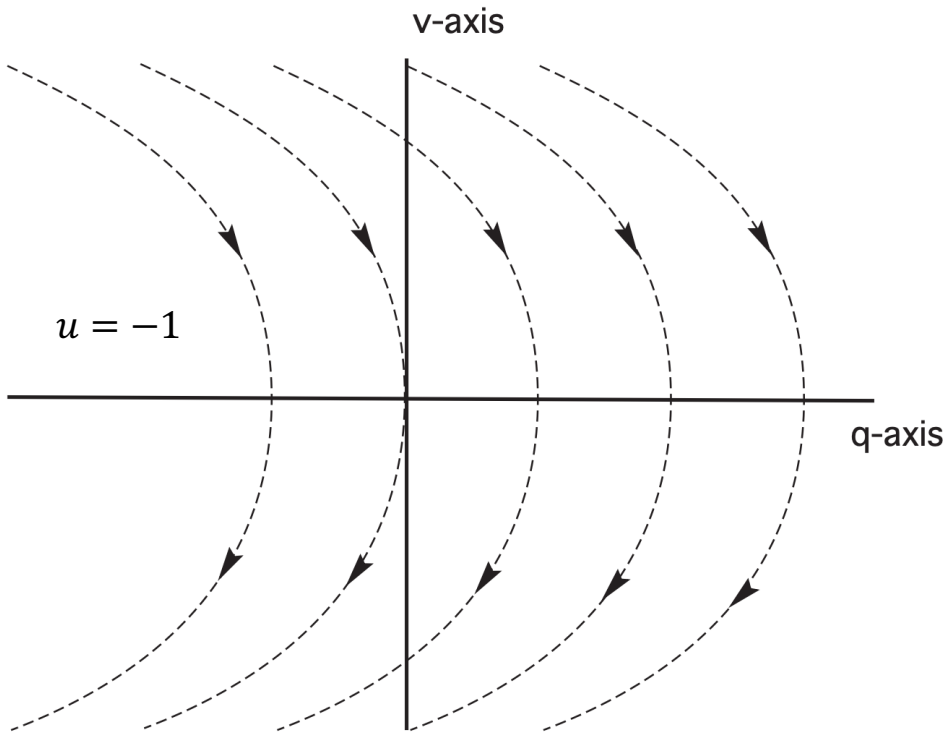
$$\begin{cases} \dot{q} = v \\ \dot{v} = -1 \end{cases} \Rightarrow v\dot{v} = -\dot{q} \Rightarrow \frac{1}{2}(\dot{v^2}) = -\dot{q}$$

- Let t_0 belong to the time interval where $u = 1$ and integrate from t_1 to t:

$$\frac{v^2(t)}{2} - \frac{v^2(t_1)}{2} = -q(t) + q(t_1) \Rightarrow v^2(t) = -2q(t) + \underbrace{(-v^2(t_1) + 2q(t_1))}_{b}$$

A Rocket Railroad Car: A Geometric Solution

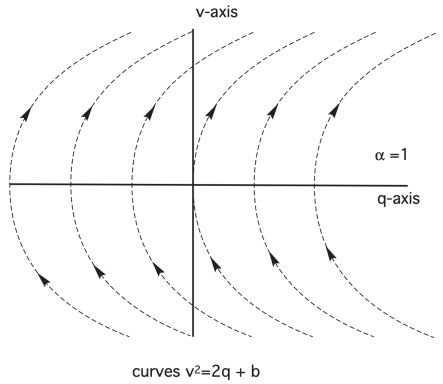
- **CASE 2**: $u = -1$ for some time interval

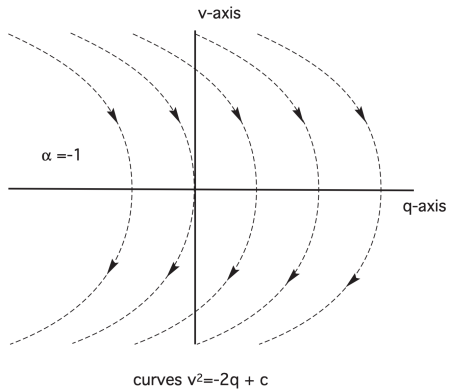


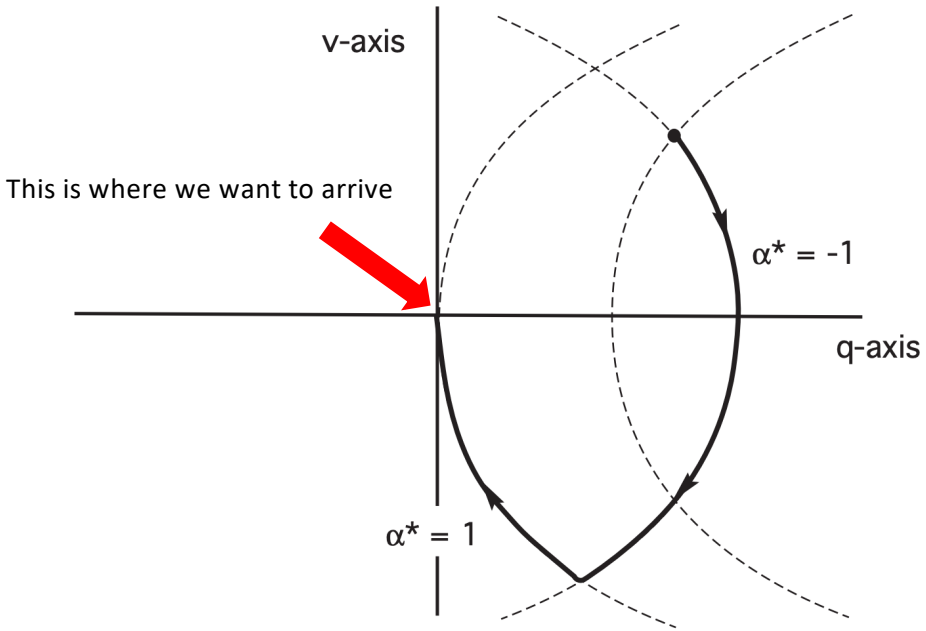
curves $v^2 = -2q + c$

A Rocket Railroad Car: A Geometric Solution

- We should reach to the origin by switching between the two parabolas back and forth





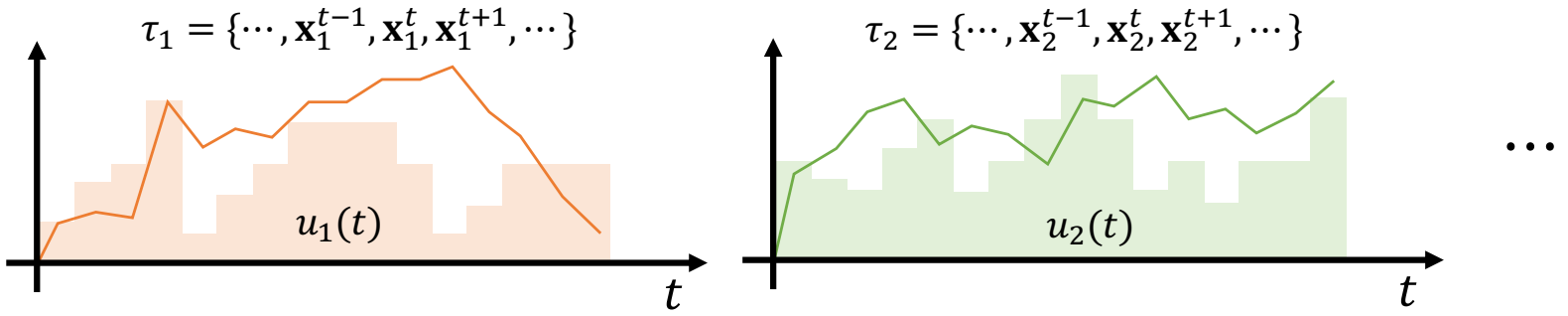


Methods for Solving Optimal Control Problems

- (Single or Multiple) Shooting
- Model Preditive Control
- Reinforcement Learning

Shooting Methods

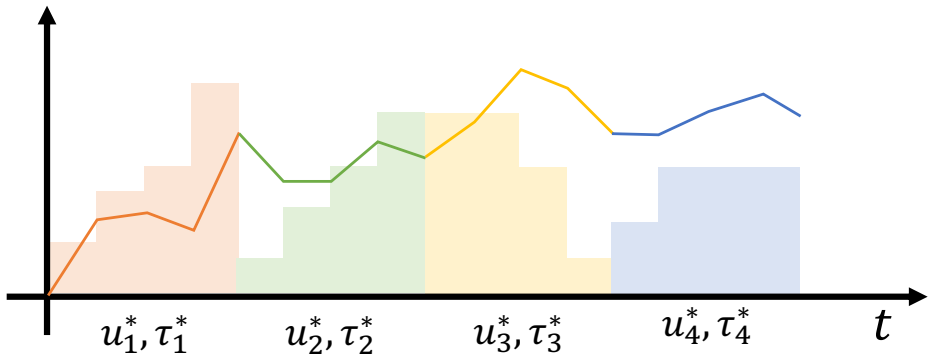
- The idea is to simply ***shoot (generate)*** many trajectories and evalute them using the payoff function, then select the best



- The simplest way to create candidate trajectories is to chose actions randomly at each timestep

Shooting Methods

- ***Single*** shooting method
  - The entire trajectory is created all at once
- ***Multiple*** shooting method
  - The entire time horizon is devided into multiple short segments and each segment is solved by a single shooting method



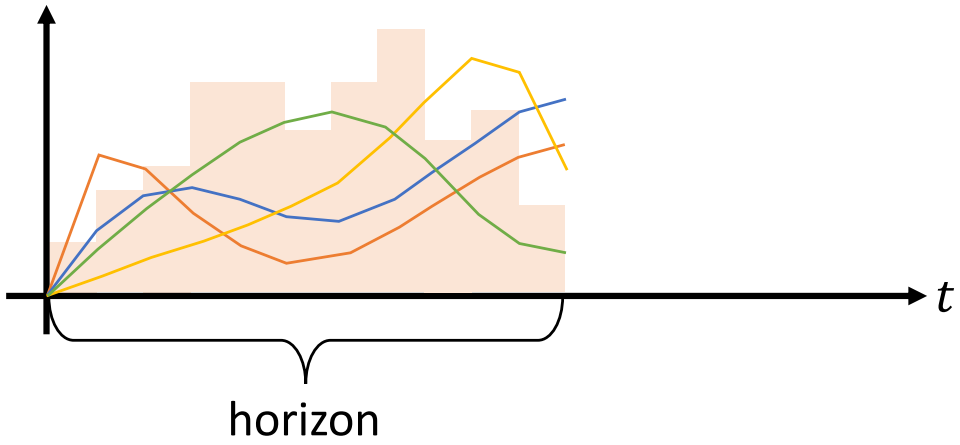
$$u^* = \{u_1^*, u_2^*, u_3^*, u_4^*\}$$
$$\tau^* = \{\tau_1^*, \tau_2^*, \tau_3^*, \tau_4^*\}$$

Shooting Methods: Examples

[Tan et al. 2014] Soft Body Locomotion

Model Predictive Control (MPC)

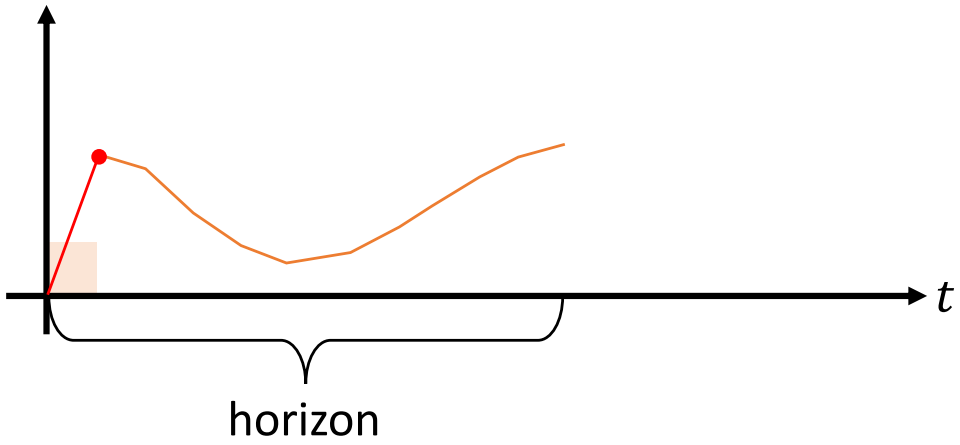
- Model predictive control (MPC) is a special case of multiple shooting methods, which decides an action for each timestep via optimization



At the first timestep, it runs a shooting method with (short) fixed time-horizon

Model Predictive Control (MPC)

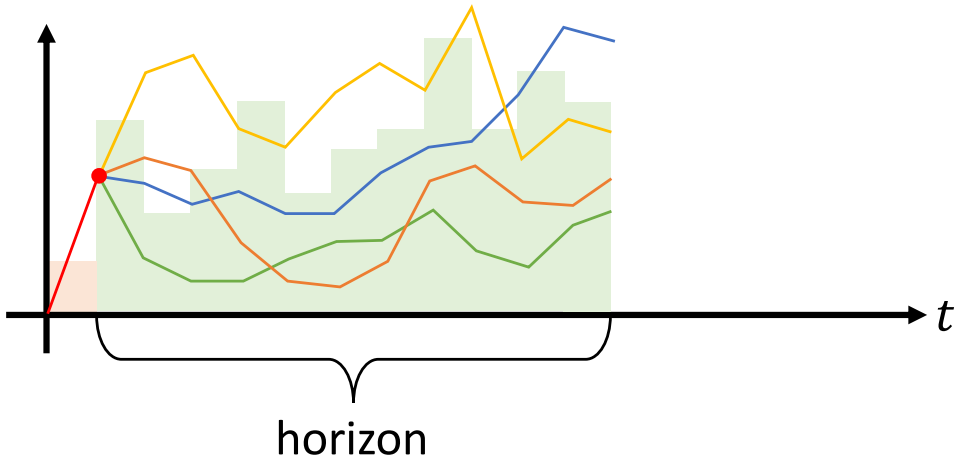
- Model predictive control (MPC) is a special case of multiple shooting methods, which decides an action for each timestep via optimization



The best one is seleted and MPC takes only the first control command $u^*(0)$

Model Predictive Control (MPC)

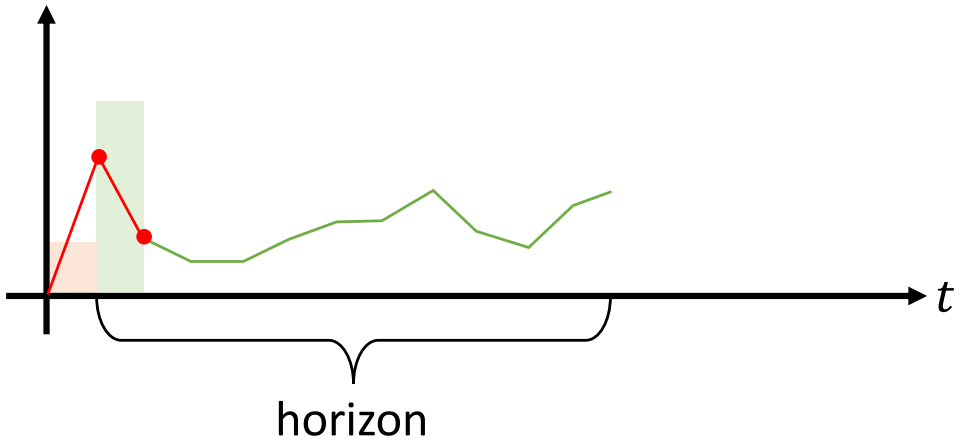
- Model predictive control (MPC) is a special case of multiple shooting methods, which decides an action for each timestep via optimization



The dynamic system is updated via $u^*(0)$, the time-horizon slides one-step forward
At the next timestep, MPC runs a single shooting method again

Model Predictive Control (MPC)

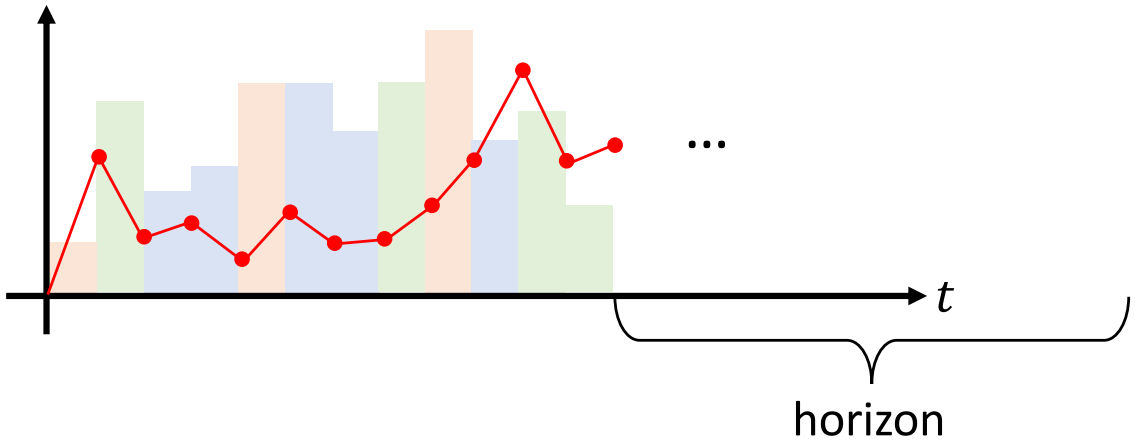
- Model predictive control (MPC) is a special case of multiple shooting methods, which decides an action for each timestep via optimization



In the same way, the best one is seleted and the first control command is only taken

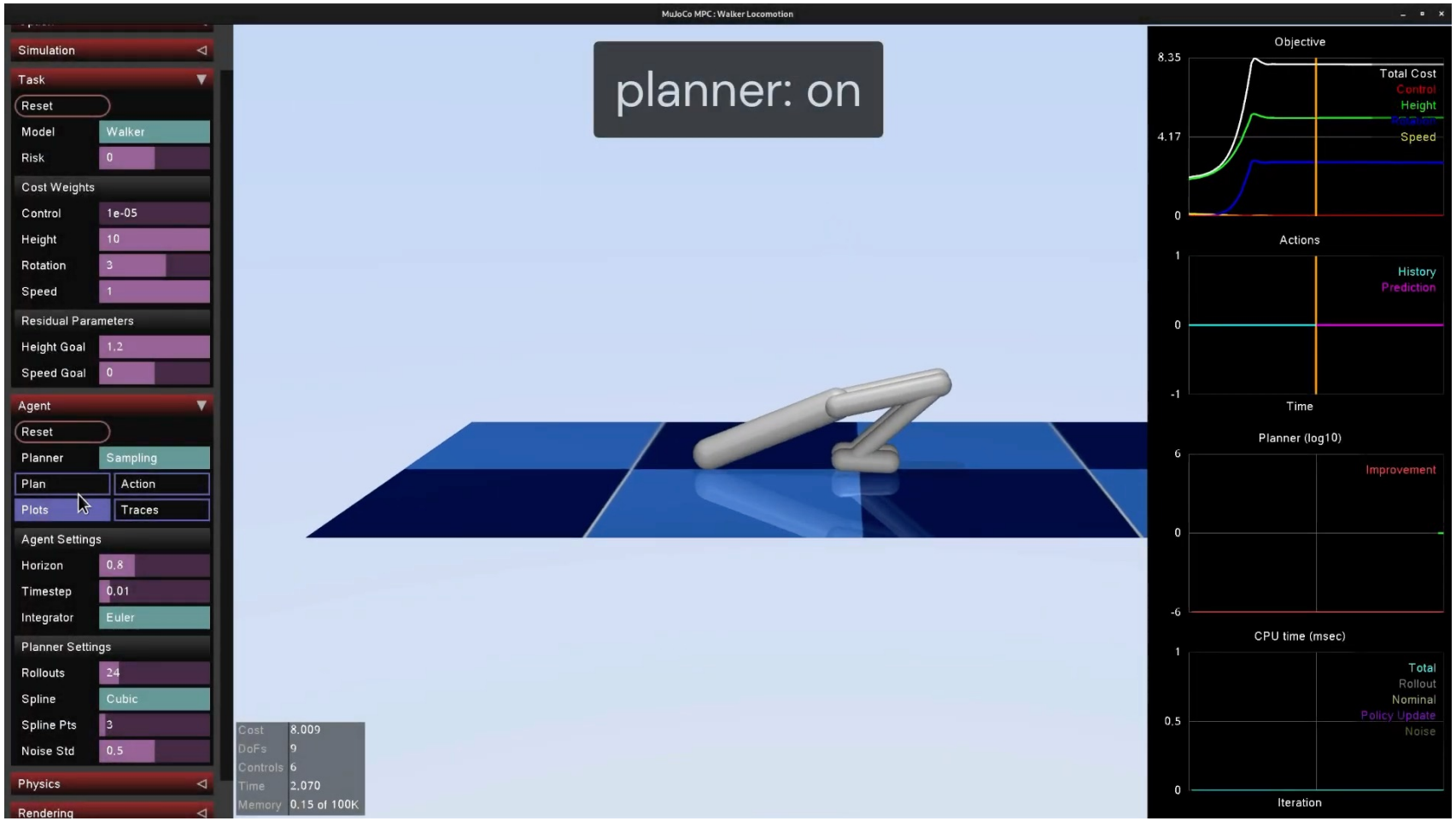
Model Predictive Control (MPC)

- Model predictive control (MPC) is a special case of multiple shooting methods, which decides an action for each timestep via optimization



This process repeats until the system ends

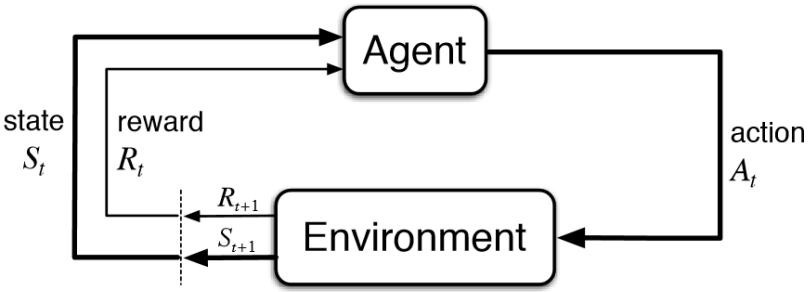
MPC: Examples



[Howell et al. 2022] Predictive Sampling: Real-time Behaviour Synthesis with MuJoCo

Reinforcement Learning (RL)

- A maching-learning approach to solve optimal control problems



reward

$$J[\mathbf{u}(\cdot)] \equiv \int_0^T r\big(\mathbf{x}(t), \mathbf{u}(t)\big)dt + g(x(T))$$

agent state action

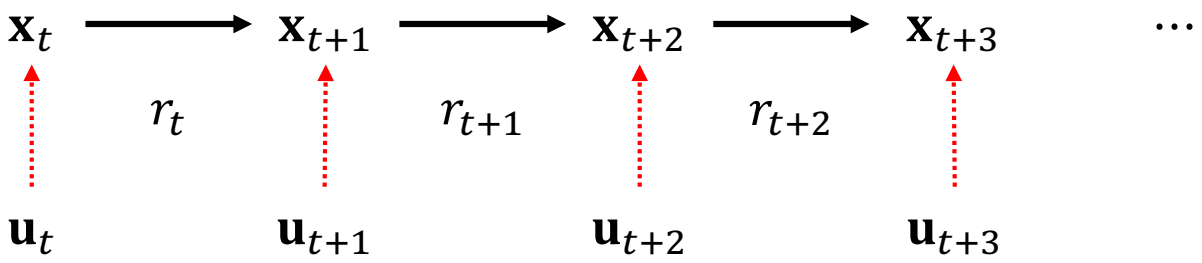
Environment

$$\begin{cases} \dot{\mathbf{x}}(t) = f\big(\mathbf{x}(t), \mathbf{u}(t)\big) \\ \mathbf{x}(0) = \mathbf{x}_0 \end{cases}$$

Markov Decision Process

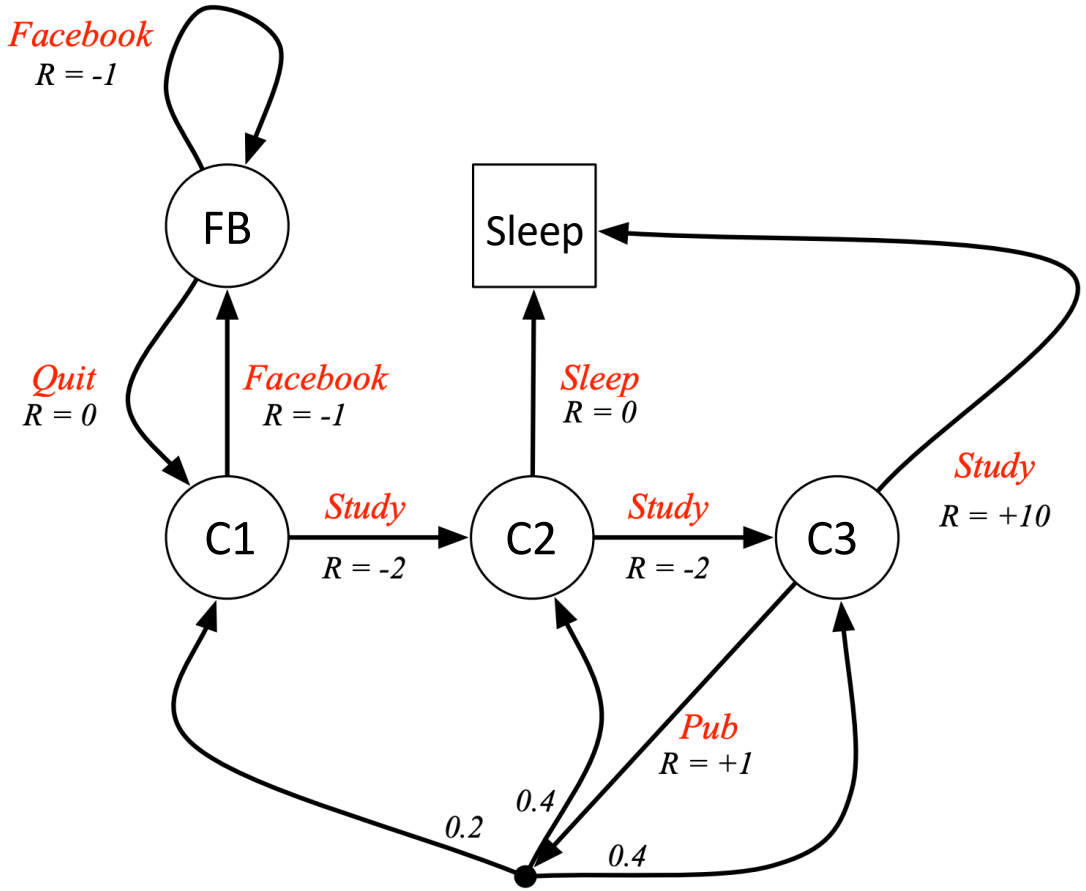
- A key idea in RL is regarding the control process as a random process, specifically **Markov Decision Process**, where the time-domain is discretized

$$\mathcal{J}[\mathbf{u}(\cdot)] \equiv \int_0^T r\big(\mathbf{x}(t), \mathbf{u}(t)\big)dt + g(x(T))$$



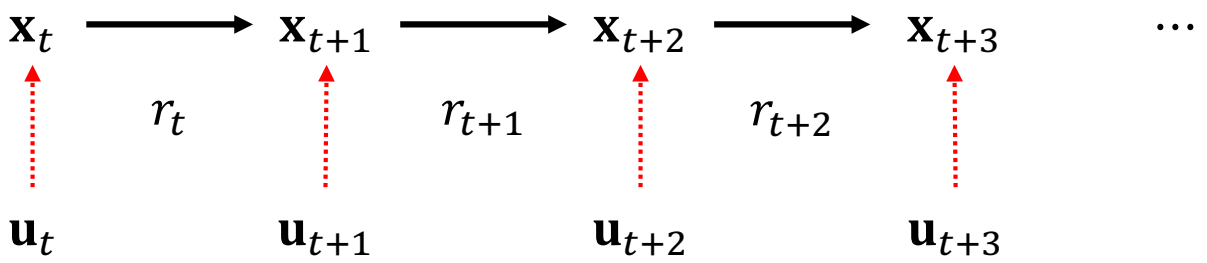
$$\mathcal{J} = r_0 + \gamma r_1 + \gamma^2 r_2 \cdots = \sum_{t=0}^{T} \gamma^t r_t$$

Student MDP


Facebook
R = -1
FB
Sleep
Quit
R = 0
Facebook
R = -1
Sleep
R = 0
Study
R = +10
C1
Study
R = -2
C2
Study
R = -2
C3
Pub
R = +1
0.4
0.2
0.4

Value Function

- Another key idea is **learning a value function** that approximates the expected return (i.e. sum of rewards)



Recursive structure

$$V_{\mathbf{u}}(\mathbf{x}_t) = \mathbb{E}_{\mathbf{u}}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} \cdots]$$
$$= \mathbb{E}_{\mathbf{u}}[r_t + \gamma(r_{t+1} + \gamma r_{t+2} + \cdots)]$$
$$= \mathbb{E}_{\mathbf{u}}[r_t + \gamma V_{\mathbf{u}}(\mathbf{x}_{t+1})]$$

Optimal Value Function

- If we control the system optimally, its value function should satisfy:

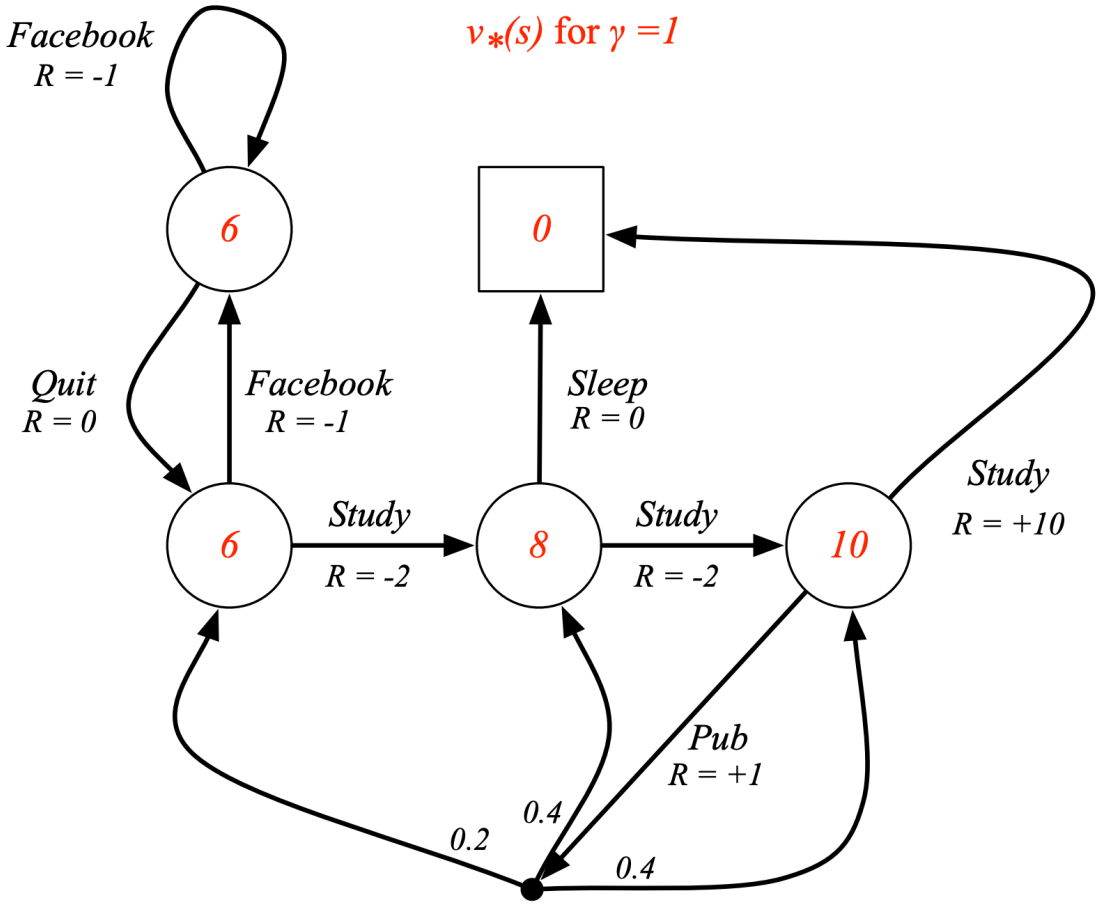
$$V_{\mathbf{u}_*}(\mathbf{x}_t) = \max\left[r_t + \gamma V_{\mathbf{u}_*}(\mathbf{x}_{t+1})\right]$$

- On the other hand, if we have an optimal value function, we can control the system optimally as follows:

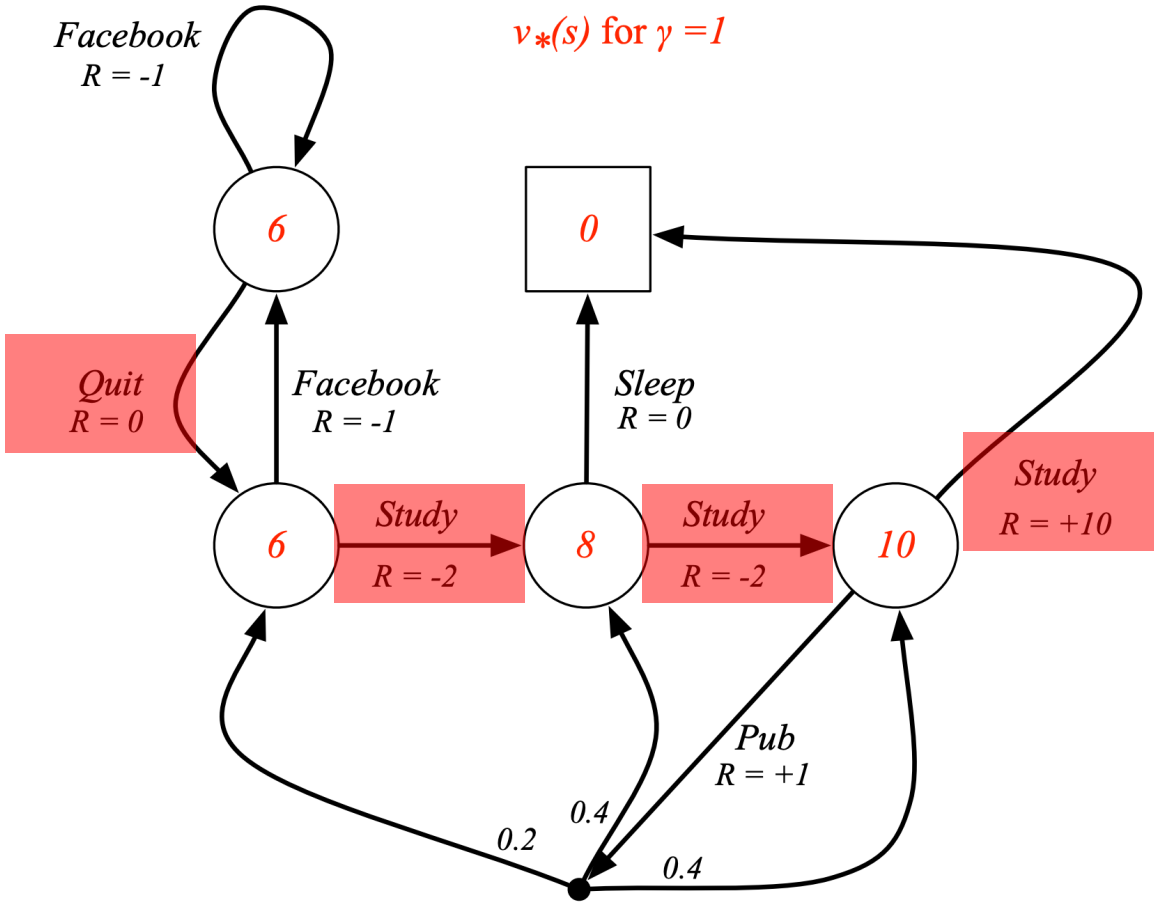
$$\mathbf{u}_*(\mathbf{x}_t) = \underset{u \in U}{\operatorname{argmax}}[r_t + \gamma V_*(\mathbf{x}_{t+1})]$$

- With a bit of exaggeration, RL is all about how to compute an optimal value function

Student MDP: Optimal Value Fn.



Student MDP: Optimal Policy



RL Algorithms: Monte-Carlo

Initialize V randomly

Repeat:

Initialize $\mathbf{x}_1$

Sample an episode $(\mathbf{x}_1, \mathbf{u}_1, r_1, \cdots, \mathbf{x}_T)$ until termination

by following a greedy policy on V

Compute **actual returns** $(\mathcal{J}_1, \cdots, \mathcal{J}_T)$

For $1, \cdots, T$

$V(\mathbf{x}_t) \leftarrow V(\mathbf{x}_t) + \alpha(\mathcal{J}_t - V(\mathbf{x}_t))$

RL Algorithms: TD(0)

Initialize V randomly

Repeat:

Initialize $\mathbf{x}_1$

Sample an episode $(\mathbf{x}_1, \mathbf{u}_1, r_1, \cdots, \mathbf{x}_H)$ with a fixed horizon H by following a greedy policy on V

For $1,2,\cdots,H$

$$V(\mathbf{x}_t) \leftarrow V(\mathbf{x}_t) + \alpha\big(r + \gamma V(\mathbf{x}_{t+1}) - V(\mathbf{x}_t)\big)$$

Monte-Carlo vs. TD(0)

- TD(0) uses biased estimate while MC doesn't

MC: $V(\mathbf{x}_t) \leftarrow V(\boldsymbol{s}_t) + \alpha\big(\mathcal{J}_t - V(\mathbf{x}_t)\big)$

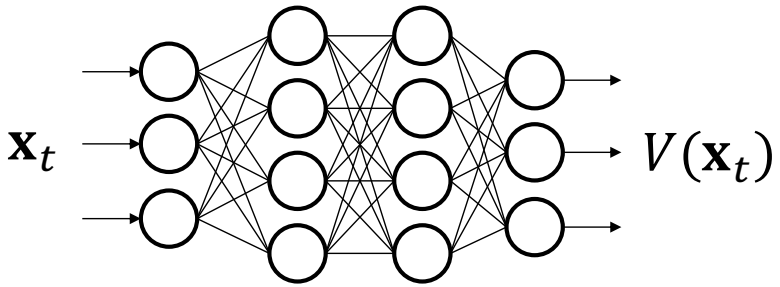
TD(0): $V(\mathbf{x}_t) \leftarrow V(\mathbf{x}_t) + \alpha\big(r_t + \gamma V(\mathbf{x}_{t+1}) - V(\mathbf{x}_t)\big)$

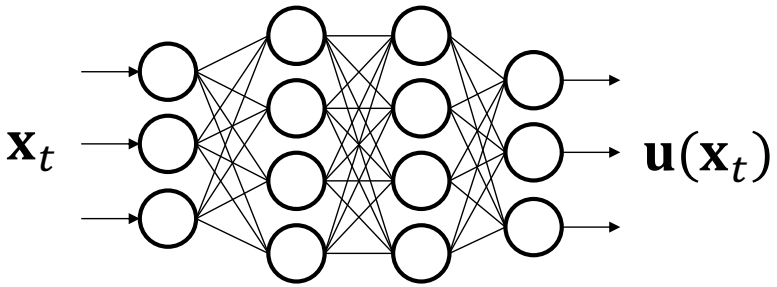
- TD(0) can learn **without** the final outcome
  - TD(0) can learn from incomplete sequences while MC can only learn from complete sequences
  - TD(0) works in continuing environments while MC only works for episodic (terminating) environments

Deep Reinforcement Learning

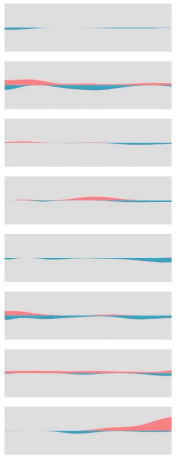
- Classical RL algorithms work on small size problems only (low-dimensional and discrete state/action)
  - In such environment, value functions and policies are represented by tables

- DeepRL uses deep neural networks to handle high-dimensional continuous state (+action)





DeepRL: Examples



[Won et al. 2022] Physics-based Character Control via Conditional VAEs, SIGGRAPH 2022

[Min et al. 2019] SoftCon: simulation and control of soft-bodied animals with biomimetic actuators